

## **Relazione Advanced programming Languages**

### **Abstract**

Il nostro progetto ha come obiettivo la realizzazione di un sistema per lo scambio di file tra host posti in una rete che utilizza la tecnica NAT.

Per la nostra condivisione abbiamo posto come prima opzione di collegamento una tecnica sperimentale per il NAT traversal detta Hole Punching.

Purtroppo, essendo una tecnica dipendente dalle singole implementazioni del NAT è soggetta a molti fallimenti, quindi abbiamo implementato allo stesso tempo un metodo alternativo di comunicazione tramite un Server Mediator in Go.

L'utente della nostra applicazione dovrà registrarsi, accedere e in seguito avrà la possibilità di stabilire quali e quanti file trasmettere ed attraverso un codice univoco associato ad ogni sessione utente si potrà tentare di accedere in ricezione ai file condivisi da un altro utente che potranno essere visualizzati e scaricati.

### **Divisione moduli e rispettivi compiti**

La nostra applicazione è composta da tre differenti moduli implementati rispettivamente nei linguaggi di programmazione C#, Python e Golang.

Il primo implementa l'interfaccia grafica necessaria a poter interagire con il sistema attraverso l'utilizzo di semplici pulsanti e textbox, allo stesso tempo si occupa di una parte di logica applicativa per quanto riguarda il signin ed il login.

Nel Form1 e nel Form2 è stata implementata la comunicazione necessaria a inviare i dati per la registrazione o accesso e ricevere le risposte dal server scritto in Golang che ha anche il compito di gestire il nostro database MySQL.

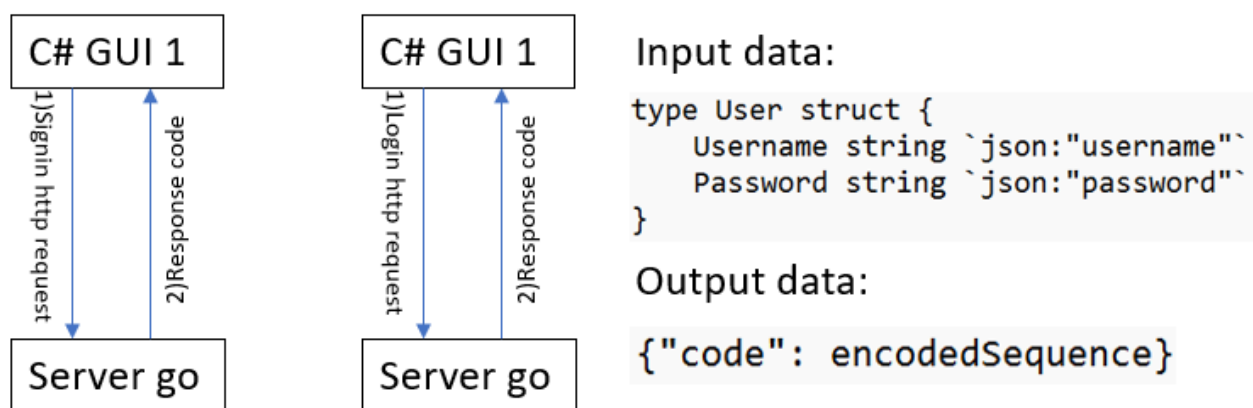
Il secondo modulo, scritto in python, si occupa di implementare tutta la logica lato client del nostro protocollo di comunicazione (eccetto la

registrazione e login) e ascolta le richieste provenienti dall'interfaccia grafica che indicano quale operazione eseguire.

Spiegheremo nella sezione "Modulo python" le varie operazioni implementate.

Il terzo modulo, scritto in Golang, si occupa di gestire il nostro database, implementa un server http con 5 route predefinite che realizzano il nostro protocollo di comunicazione lato server, inoltre, gestisce un server basato su socket TCP sulla porta 5000 che realizza lo scambio di messaggi e richieste necessarie per l'implementazione dell'Hole Punching.

## Modulo C#

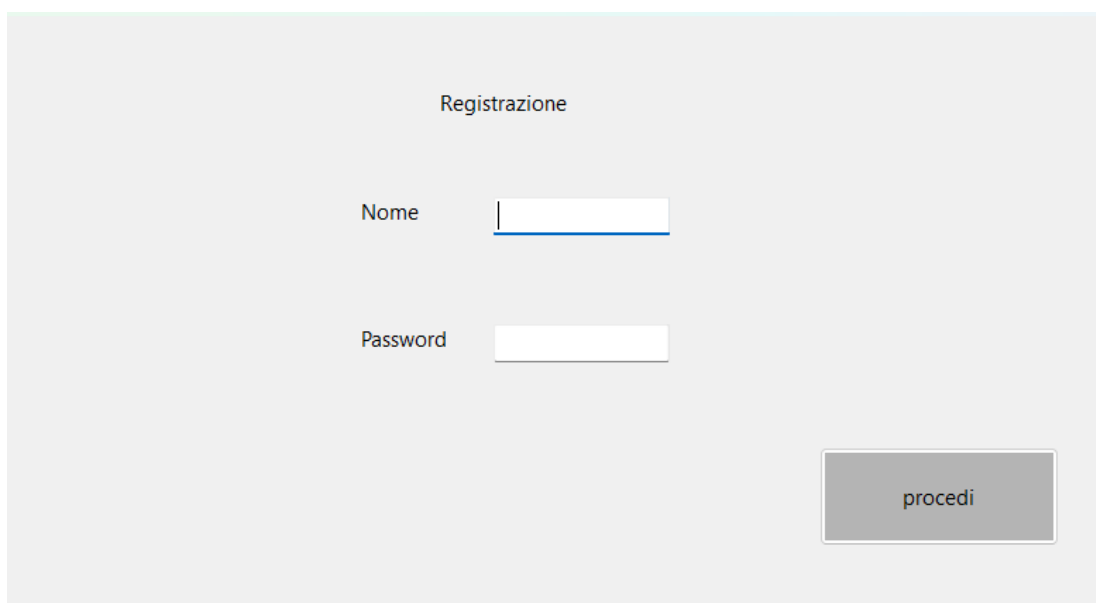


Il modulo in C# si articola in 4 form diversi: il primo, visibile nella figura sotto, si occupa del login dell'utente, ed è diviso in due sezioni, nella sezione a sinistra si trovano due TextBox che l'utente dovrà compilare e un pulsante di conferma, l'evento del click sul pulsante se i due campi sopra di esso non sono vuoti farà partire una richiesta http post al server go sulla route "/login" e se il login va a buon fine il client avrà come risposta un codice alfanumerico di 4 cifre che rappresenterà il codice dell'utente per questa sessione e di conseguenza si aprirà il form relativo alla trasmissione. Se non sono state inserite le credenziali verrà resa visibile una label contenente il testo "inserire credenziali".



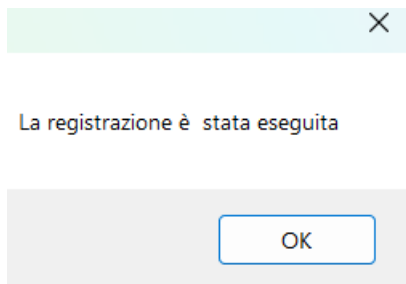
The image shows a login form titled "Login" centered at the top. Below the title, there are two input fields: "Nome" (Name) and "Password". To the right of the "Nome" field, there is a text link that says "Se vuoi registrarti clicca sotto" (If you want to register click below). Below the "Password" field, there is a blue button labeled "ACCEDI" (Log In). To the right of the "Password" field, there is a button labeled "REGISTRATI" (Register).

Nella parte a destra invece troviamo un pulsante che consente di passare al form relativo alla registrazione, il click su di esso farà apparire il suddetto form rendendo il precedente non usabile finché non verrà chiuso quello appena aperto.

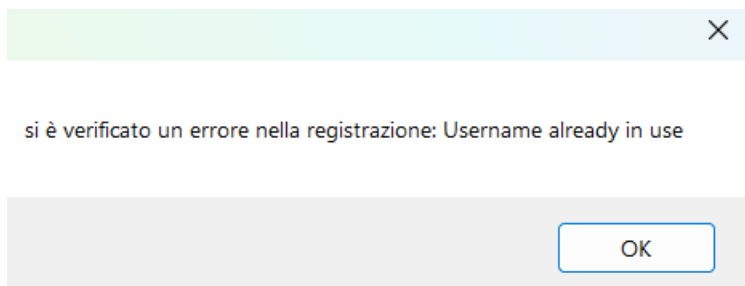


The image shows a registration form titled "Registrazione" centered at the top. Below the title, there are two input fields: "Nome" (Name) and "Password". Below the "Password" field, there is a gray button labeled "procedi" (Proceed).

In questo form al pulsante procedi è legato un evento onClick che se non sono stati riscontrati errori nei campi fa partire una richiesta post alla route `"/signin"` del server go che risponderà con un codice se è andato tutto bene e ciò si traduce lato client in un messageBox, che è riportato sotto, successivamente ci sarà la chiusura di questo form in modo da poter procedere con il login.



La richiesta post non verrà inviata se lato client si riscontrano degli errori nei campi, tali errori sono: entrambi i campi vuoti, almeno un campo vuoto e password con una lunghezza minore di 7 caratteri. Se si verifica una di queste condizioni verrà resa visibile una label che notifica l'errore. Se invece non si verifica nessun errore lato client e la richiesta viene inviata al server oltre alla risposta positiva già discussa prima ci potrebbe essere una risposta negativa dato dal fatto che esiste già un utente registrato con quel dato username, in questo caso la risposta alla richiesta post notifica la cosa e lato client viene mostrato il seguente messageBox.



Ora analizziamo il terzo form che si apre se il login va a buon fine, l'apertura di questo form innesca l'esecuzione del modulo python e una richiesta specifica verrà inoltrata a python, tale richiesta è caratterizzata da un campo denominato query che assumerà il valore "start\_share" che identifica il tipo di operazione. Il Form in questione è riportato nella seguente immagine.

inserisci i file da rendere visibili

Cambia Visualizzazione

il tuo username =







utente1

il tuo codice =

txBf

rimuovi

Aggiungi

Name	Path
 1.pdf	C:\Users\giuse\OneDrive\Documenti\cose\1.pdf
 iot	D:\uni\iot
 CD_Lezione9_II_PCM.pdf	C:\Users\giuse\OneDrive\Documenti\GitHub\Hole-...
 oregairu.png	C:\Users\giuse\OneDrive\Immagini\oregairu.png
 tds.docx	C:\Users\giuse\OneDrive\Documenti\tds.docx
 1Blockchain.mp4	D:\uni\iot\Blockchain 2021 novembre-20231212T1...

Aggiungi Cartella

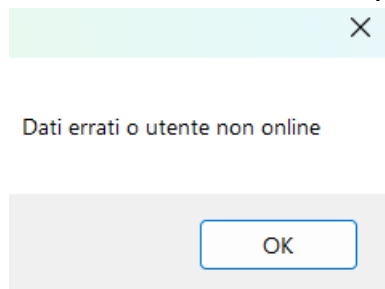
iinserire username

Inserire il codice

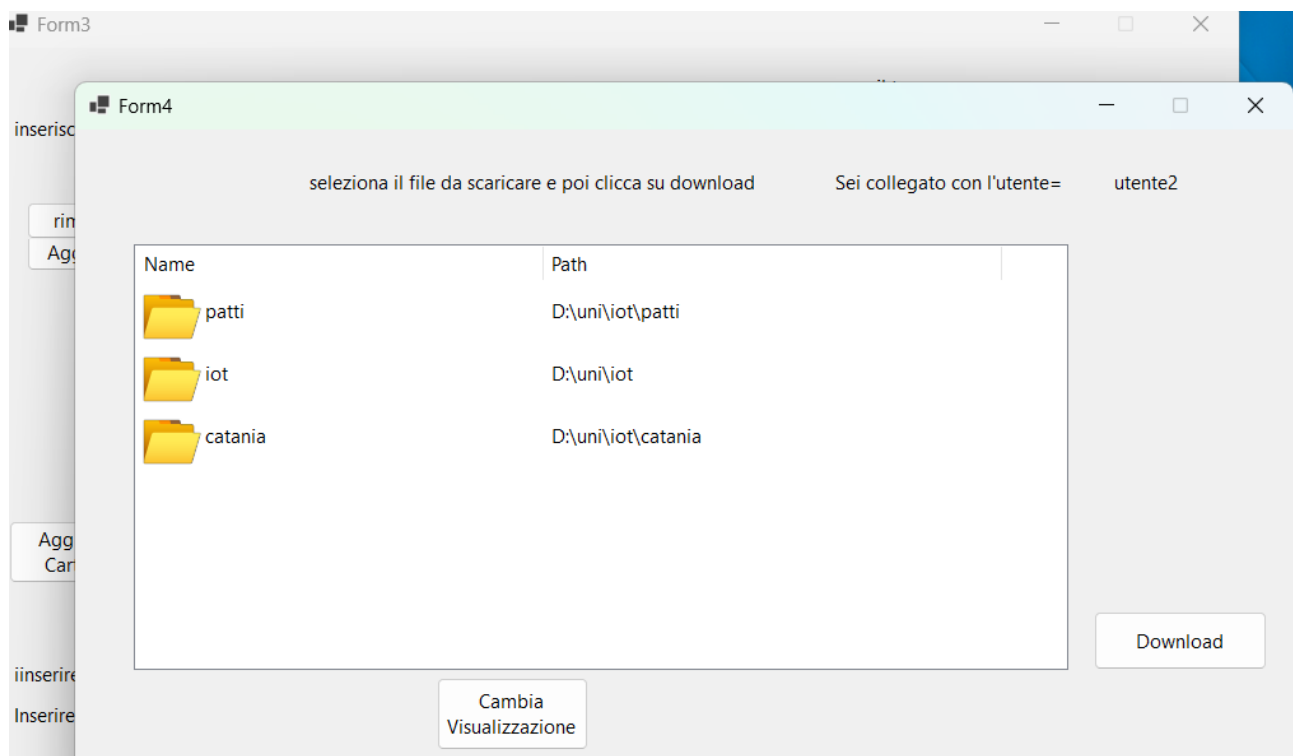
Invio

In alto a destra troviamo un riepilogo delle due credenziali che dovranno essere comunicate in separate sede ad un altro utente per poter effettuare la connessione, cioè il proprio username e il codice di sessione generato al momento del login. In alto al centro si trova il pulsante “Cambia visualizzazione” il cui click permette il cambio di visualizzazione della ListView, in particolare ci sono tre tipi di visualizzazione “dettaglio”, “Small Icon” e “Large Icon”. Al centro del Form si trova la listView che con l’apertura del form si riempirà con i precedenti file scelti dall’utente in una sessione precedente se è avvenuta. A sinistra si trovano tre pulsanti tutti relativi all’interazione con la ListView. Il Primo pulsante “rimuovi” rimuove l’elemento selezionato dalla lista, il pulsante “aggiungi” permette all’utente di inserire un file mentre “aggiungi Cartella” permette l’inserimento di una cartella intera. Tutti e tre questi pulsanti innescano una richiesta a python sempre con valore “start\_share” in quanto modificano l’elenco degli elementi che si è deciso di poter condividere. Infine in fondo a sinistra della pagina sono presenti due campi da editare ed un pulsante per l’invio, questa sezione permette di poter tentare di collegarsi ad un altro utente specificando il suo username e il suo codice di sessione, viene fatto un semplice controllo sul campo relativo al nome in quanto non ci si può collegare con se stessi e inoltre si verifica anche che entrambi i campi non siano vuoti, se non c’è nessuno di questi errori viene inoltrata una richiesta a python specificando come operazione “names” in quanto si è interessati

a ottenere la visualizzazione degli elementi condivisi dall'utente specificato. Potrà esserci una risposta negativa segnalata dal seguente messageBox:

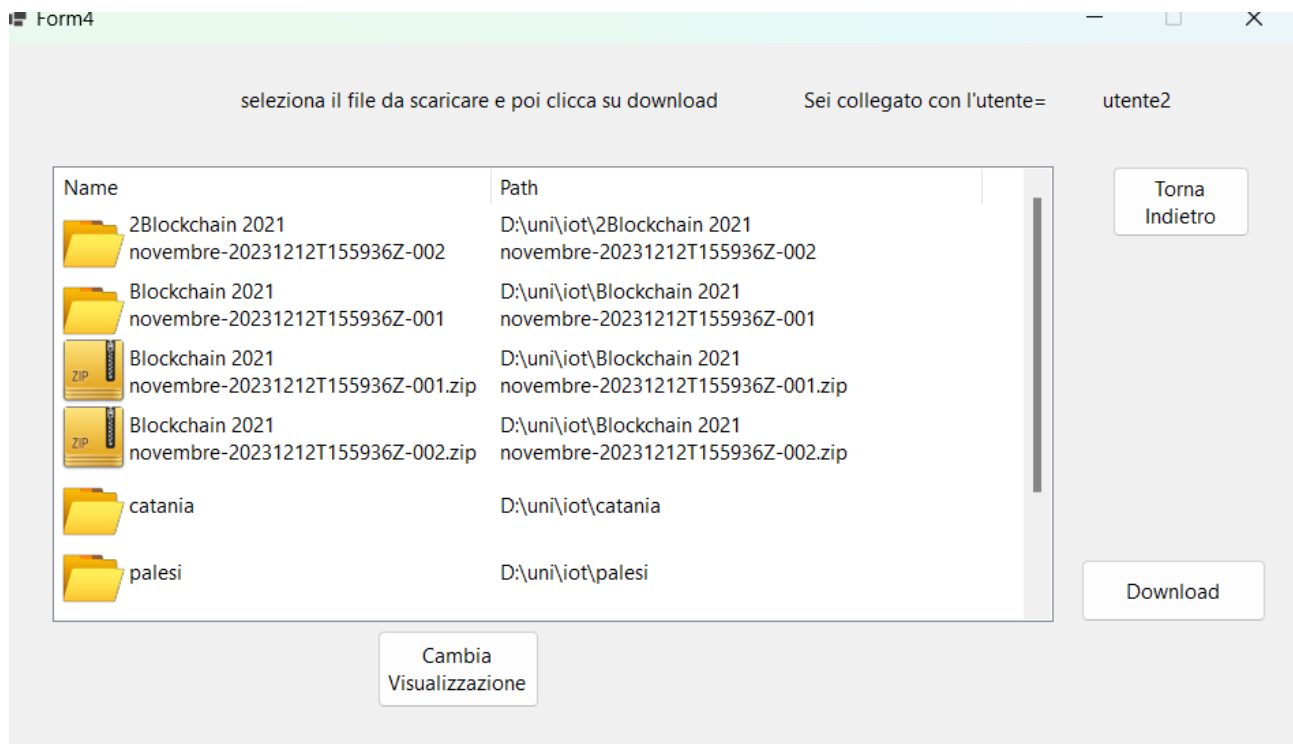


Oppure una risposta affermativa che comporta l'apertura del quarto form che mostra i file che l'utente a cui ti sei collegato ha deciso di rendere visibili. In seguito, è mostrato il quarto form relativo alla ricezione.

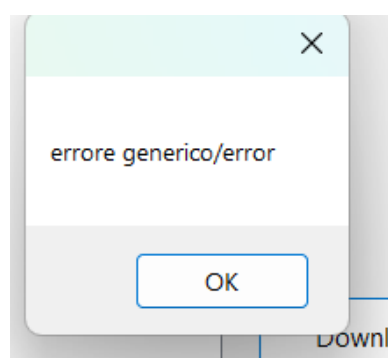
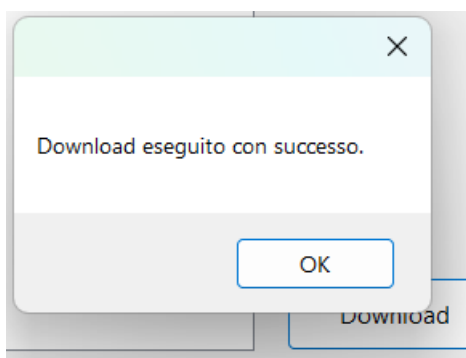


Questo form ha una struttura molto simile al form di trasmissione, infatti, anche esso presenta il pulsante per cambiare la visualizzazione, inoltre presenta un pulsante che consente il download di un file/cartella selezionato. Inoltre, in questo form è previsto un evento di doppio click sugli elementi della listView, in particolare sulle cartelle che comporterà l'invio di un ulteriore richiesta a python sempre di tipologia "names" ma stavolta specificando il path della cartella selezionata. Ciò permetterà l'apertura di un ulteriore form di tipologia ricezione con al cui all'interno il contenuto della cartella selezionata, inoltre in questi form di ricezioni aperti

dall'evento double Click su cartella sarà previsto un ulteriore pulsante "Torna Indietro" che consentirà di tornare a visualizzare il form di ricezione antecedente.



La chiusura mediante X in alto a destra causerà la chiusura di tutti i form di ricezione aperti in modo tale da poter tornare ad interagire con il form di trasmissione. Selezionando un qualsiasi elemento della listView e cliccando sul pulsante Download partirà una richiesta a python, lato client si riceverà un messageBox che comunicherà la riuscita dell'operazione oppure un messaggio di errore.



## Modulo python

Il modulo scritto in python è composto da due file:

1. tcp\_client.py

## 2. utils.py

Il primo utilizza il framework “Flask” per implementare un’interfaccia http che risponde solo alle richieste locali (inviate dall’interfaccia grafica).

Esso implementa la route “/” di tipo Post differenziando le operazioni richieste tramite un campo “query”.

```
Username      string `json:"username"`  
Code          string `json:"code"`  
Peer_username string `json:"peer_username"`  
Peer_code     string `json:"peer_code"`  
Query         string `json:"query"`  
Path          string `json:"path"`
```

*Rappresentazione della struttura del json in arrivo*

Tipologie di query:

1. **“names”**: ritorna i nomi di tutti i file che si trovano sotto il path contenuto nel campo “path” (se gli altri campi sono corretti).
2. **“download”**: tramite lo stesso campo “path”, visto nella query precedente, ritorna il file richiesto (in binario) o ritorna l’intera directory con tutti le sue subdirectory e file.
3. **“start\_share”**: riceve tutti i path che l’utente sta condividendo come lista contenuta nel campo “path” e attiva un client che tramite un polling ciclico sul server accetta le richieste in arrivo per l’utente (se l’username e il code inserito dai richiedenti sono corretti).

### Implementazione di una richiesta:

Come primo punto sottolineiamo che una richiesta può essere eseguita dal nostro sistema o dopo aver aperto un canale tramite l’hole punching o utilizzando il server in go come mediatore tra le richieste tra i due client.

Puntualizziamo che comunque per aprire un canale si utilizza sempre una richiesta al server http per verificare i dati degli utenti e successivamente si passa alla comunicazione tramite socket TCP per lo scambio degli indirizzi IP dei due client che desiderano comunicare tra loro.

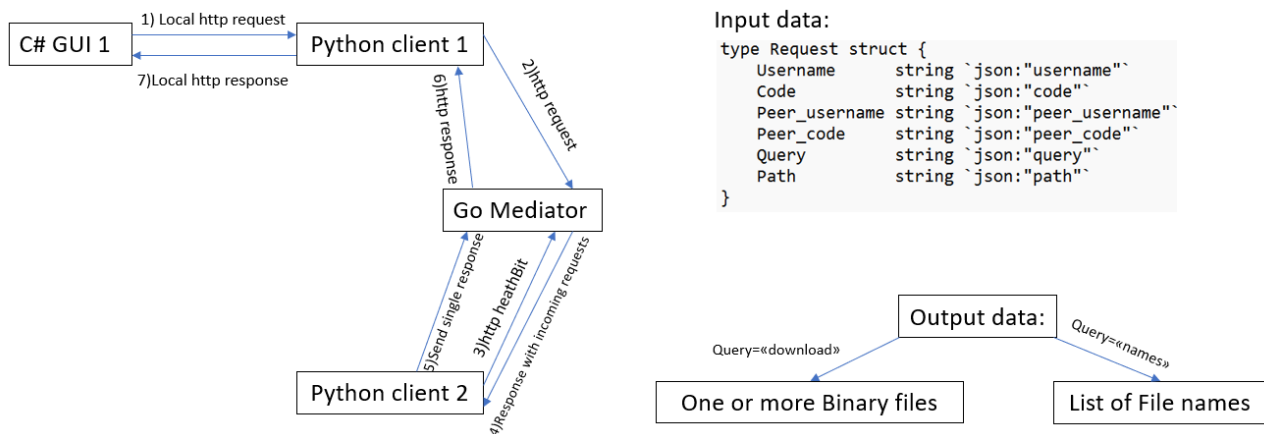
Nel file utils.py abbiamo implementato due classi e alcune funzioni ausiliarie necessarie per lo svolgimento delle operazioni intermedie che



gestiscono l'invio dei messaggi tramite il server mediator, la loro ricezione e la creazione di un canale tramite l'hole punching e la gestione dell'eventuale fallimento.

## Request

http:



La richiesta http dopo essere stata inviata al server non può essere semplicemente essere consegnata al secondo client, poiché essendo esso sotto rete che utilizza la tecnica NAT non è semplicemente contattabile, quindi sarà il secondo client a contattare il server ciclicamente per assicurarsi che ci siano richieste pendenti a suo nome e per comunicare che esso accetta richieste (stato di online). Chiameremo l'operazione precedente HeathBit.

Le richieste in arrivo hanno in esse un codice univoco che le individua e permette al secondo client di inviare una successiva richiesta di Response http con la risposta (contenente o un file binario o una serie di path), identificabile in modo da essere forwardata al client 1.

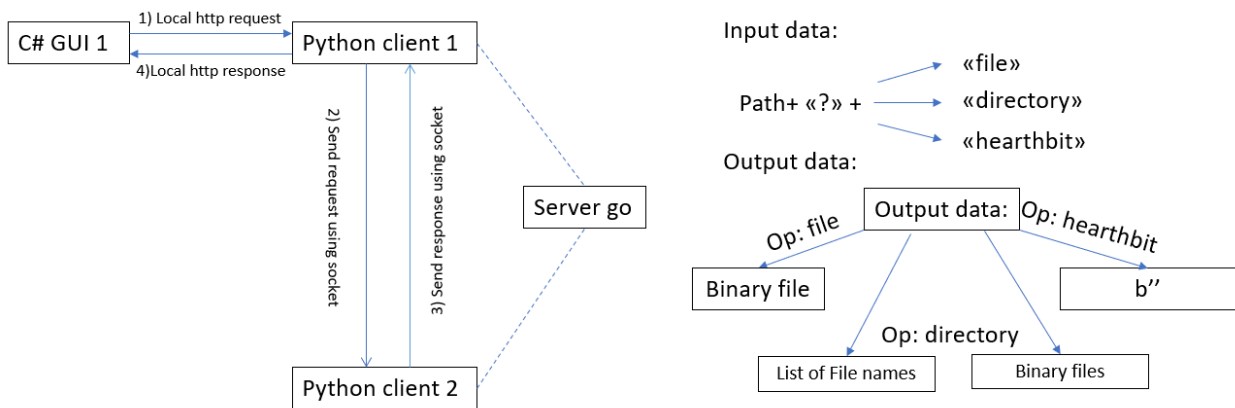
L'intera operazione è rappresentata in figura sopra.

## Hole punch Protocol:

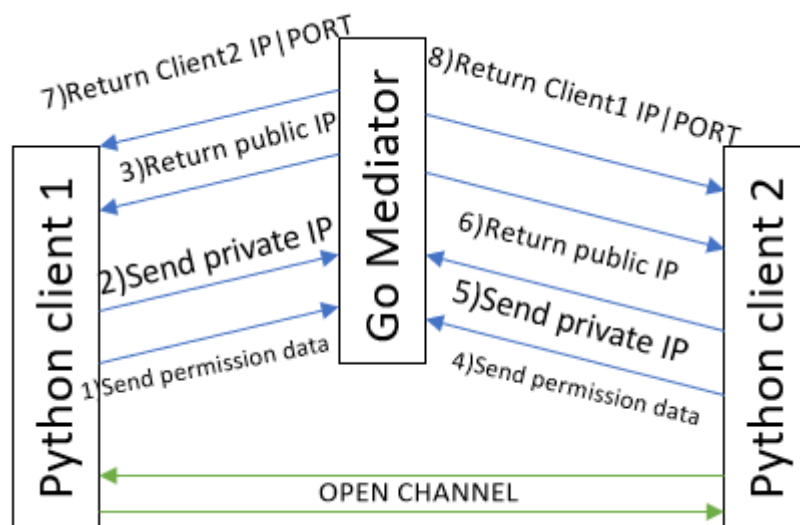
Durante la gestione delle operazioni "names" e "download" per prima cosa si controlla se esiste un canale di comunicazione esistente con un client e se esiste si inviano direttamente delle richieste tramite socket come mostrato in figura sotto.

Nelle richieste , gestite in modo minimale si invia semplicemente il path richiesto e un comando che rappresenta il download o la richiesta dei nomi dei file .

Inoltre ogni 2 secondi viene inviato un messaggio di hearthbit per mantenere la connessione attiva.



Se non esiste un canale si prova a crearlo tramite il protocollo descritto nella figura in seguito.



Dopo aver inviato una Request http per ottenere i permessi (come descritto nella sezione precedente) e dopo aver ottenuto una risposta valida si inizia la comunicazione con il mediatore go che si occupa di scambiare tra i due client i rispettivi IP come stringhe (IP privato:porta privata| IP pubblico:porta pubblica)

A questo punto ogni client avvia 4 thread che cercano di ricevere o inviare ad ognuno dei due indirizzi arrivati in precedenza.

Se non ricevono risposta entro 5 secondi la comunicazione avverrà attraverso il mediatore Go e non più tramite il canale privato.

## Modulo go

Il modulo scritto in Golang implementa un server che risponde a 5 diverse route http:

1. “/signin”
2. “/login”
3. “/hearthbit”
4. “/request”
5. “/response”

Allo stesso tempo implementa un server basato su semplice socket TCP sulla porta 5000 che implementa le operazioni descritte precedentemente nel modulo python per l’implementazione del protocollo necessario per l’hole punching.

**La prima route** si occupa dell’operazione di signin, riceve un json contenente un username ed una password, controlla se sono stati inseriti caratteri non validi e controlla se nel database esiste il nome utente con la rispettiva password.

```
CREATE TABLE IF NOT EXISTS users (  
  username VARCHAR(255) PRIMARY KEY,  
  password VARCHAR(255) NOT NULL,  
  code VARCHAR(10) ,  
  last_hearthbit TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Se supera tutti i test crea un user nella tabella users nel DB e ritorna un codice di sessione come risposta.

Sennò ritorna un json di errore.

**La seconda route** riceve sempre un json contenente un username ed una password, controlla se sono stati inseriti caratteri non validi e controlla se nel database esiste il nome utente con la rispettiva password.

Se supera tutti i test ritorna un codice di sessione come risposta per notificare al client il successo dell'operazione.

Sennò ritorna un json di errore.

**La terza route** riceve un json contenente un username ed un codice di sessione, controlla se sono stati inseriti caratteri non validi e controlla se nel database esiste il nome utente ed il rispettivo codice di sessione.

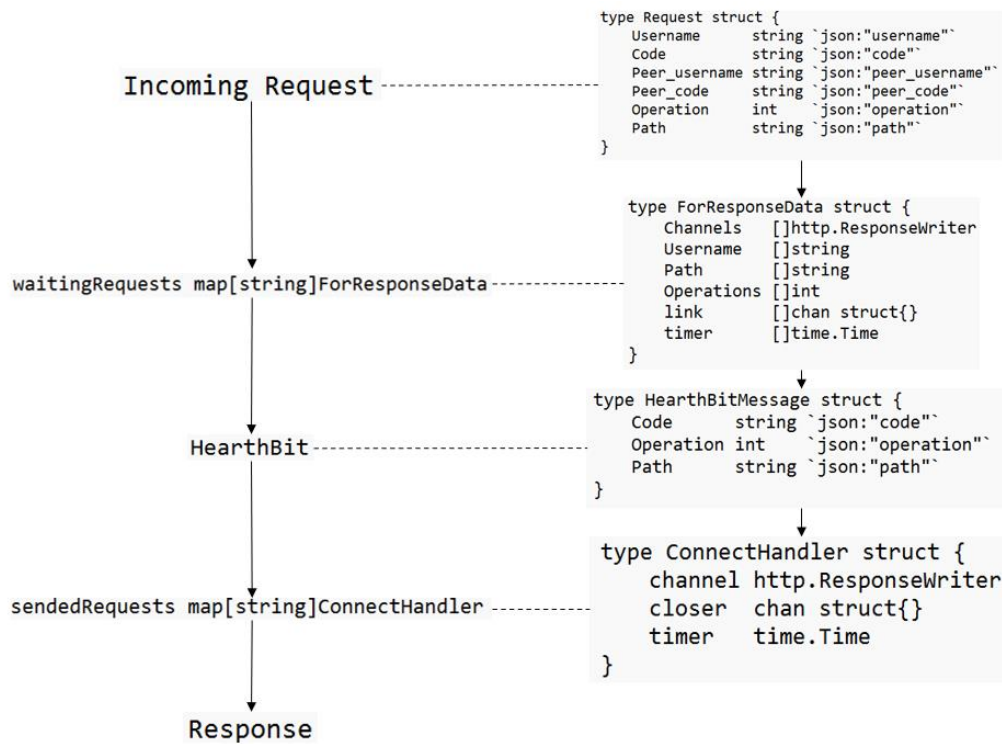
Se supera tutti i test aggiorna il parametro last\_hearthbit dell'utente inserito a NOW () e ritorna tutte le richieste in attesa del client.

Sennò ritorna un json di errore.

**La quarta route** riceve un json contenente due username e due codici di sessione, uno per il richiedente ed uno per il gestore della richiesta, oltre che un parametro "operation" ed un "path", controlla se sono stati inseriti caratteri non validi e controlla se nel database esistono i nomi utente ed i rispettivi codici di sessione.

Se supera i test inserisce la richiesta in una struttura contenente tutte le richieste in attesa del gestore della richiesta con tutti i dati necessari per la risposta.

**La quinta route** riceve un messaggio da forwardare e inserisce come parte finale dell'URL della route un parametro code che corrisponde ad un codice fornito durante la fase di ricezione delle richieste nell'Handler dell'Hearthbit in modo da poter riconoscere la singola richiesta e ritornare il risultato alla corretta connessione pendente.



Inoltre abbiamo implementato un sistema di pulizia delle strutture dati, chiamato “garbageCollector”, basate su timer allo scopo di rimuovere le connessioni pendenti senza risposta